REMARKS/ARGUMENTS

Claims 1-42 are pending in the present application. Claims 2, 5, and 8 have been amended and claims 34-42 have been canceled. Reconsideration of the claims is respectfully requested.

I. 35 U.S.C. § 102, Anticipation

I.A. Claims 1-9

The Examiner has rejected claims 1, 4, and 7 under 35 U.S.C. § 102(b) as being anticipated by *Moughani*, Data Processing System Having a Trace Mechanism and Method Therefor, United States Patent No. 5,970,246, (issued, October 19, 1999) (hereinafter, "*Moughani*"). This rejection is respectfully traversed.

The Examiner has rejected these claims stating:

> As per claim 1, Moughani discloses a method for reducing interrupts while tracing an application in a data processing system, the method comprising: receiving at a tracing function an indication that at least a portion of executable code from an application has been loaded into a memory block (Moughani, col. 4, lines 5-34, where the user accesses memory that must be valid and verified, meaning the memory has loaded executable code) prior to execution of the portion of executable code (Moughani, col. 4, lines 35-40, where the access is made with the intent to execute, but the trace bit causes the access to first generate a trace); and altering by the tracing function at least one operating-system-defined memory access protection parameter to allow read access to the memory block (Moughani, col. 3, line 51, through col. 4, line 4, where the altering is done by the setting of the trace bit).

Office Action dated November 17, 2004, p. 2.

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-Clark Corp.*, 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). In this case, each and every feature of the presently claimed invention is not identically shown in the cited reference, arranged as they are in the claims.

Claim 1 recites:

> 1. A method for reducing interrupts while tracing an application in a data processing system, the method comprising:

> receiving at a tracing function an indication that at least a portion of executable code from an application has been loaded into a memory block prior to execution of the portion of executable code; and
>
> altering by the tracing function at least one operating-system-defined memory access protection parameter to allow read access to the memory block.

In the present case, each and every step in claim 1 is not shown in the cited reference. *Moughani* does not teach the receiving and altering steps in the manner recited in claim 1. The Examiner cites the following section from *Moughani* as teaching this feature of the claimed invention.

> At decision block 70, it is determined if the address for the particular memory segment, such as a page of memory, to be accessed is valid by checking the valid bit in, for example, bit field 54 of FIG. 5. A valid page is a page, or memory segment, that has been defined by the program. Each page or memory segment corresponds to a memory or peripheral area, and has a valid bit field associated with it. Once the information of a page of memory is defined, the information located in that page or memory segment is valid for use by the system. The valid bit is then asserted to indicate that the information in that page is defined and valid. If an access to a page that is not valid (i.e. the valid bit is negated), the NO path is taken to step 72 and an interrupt is generated, and an interrupt handling routine is to be executed. The interrupt may, for example, cause data processing system to shut down, or provide predetermined information to a display. If the access is to a valid page, the YES path is taken and the process flow continues to decision step 74. At decision step 74, it is determined if the access attributes of the memory segment to be accessed match with the access attributes stored in access attribute register 34. In one embodiment of the present invention, illustrated in FIG. 4, this information is located in access protection field 52 of register 27. If the access attributes do not match then the NO path is taken to step 76 and an interrupt is generated. An interrupt handling routine is executed to deal with the access attribute mismatch. If the access attributes match, or compare favorably with each other, then the YES path is taken from decision step 74 to decision step 78.

Moughani, col. 4, ll. 5-34.

In this section, *Moughani* describes that an area of a memory is valid for access when a valid bit is set. Once the area of the memory is valid, an operating system or a user may access the area for the information stored therein. If the valid bit is not set for the area of the memory being accessed, an interrupt is thrown, which is handled by an interrupt handling routine.

This section fails to teach the receiving step of claim 1. The indication as recited in claim 1 recites that at least a portion of executable code from an application has been loaded into a memory block. Accordingly, the indication as recited in claim 1 indicates that some code is loaded into the memory. In contrast, *Moughani*'s section cited by the Examiner teaches that the indication in *Moughani* indicates whether the area of the memory may be accessed. A code has to be loaded into the memory before the code can be accessed. Therefore, the loading and accessing steps with respect to a memory operation are

not one and the same. Consequently, the cited section fails to teach the receiving step of claim 1 for this reason. For anticipation, every element of a claimed invention must be identically shown in a single reference, arranged as they are in the claims. *In re Bond.* The cited section of *Moughani*, and the remainder of *Moughani*'s disclosure, fail to teach this feature of the receiving step as recited in claim 1.

Claim 1 further recites receiving at a tracing function an indication of the nature as described above. Receiving the indication of the validity of the memory being accessed at the tracing function is not shown in the cited section. The section from *Moughani* only shows that the indication of the validity of the memory area is used for the memory access by the operating system or the user. A tracing function is not described in the *Moughani* disclosure as being the same as the operating system or the user. Persons of ordinary skill in the art know that a function, such as a tracing function, is a part of an application that may run under an operating system, but is not the operating system itself. This knowledge can be inferred because the mere fact that a tracing function even under the broadest possible interpretation does not contain several critical operating system functions, such as I/O management, peripherals management, and kernel functions. Therefore, the cited section fails to teach the receiving step of claim 1 for this additional reason.

As to the altering step, claim 1 recites that the altering is done by the tracing function. The Examiner cites the following section from *Moughani* to find a teaching for this feature:

> FIG. 6 illustrates a flow chart for a method of memory segment tracing of an access protected memory in accordance with the present invention. In FIG. 6 data processing system 10 is first powered up or reset in a supervisor mode. Supervisor mode indicates that the supervisor has access to all areas of the data processing system. This includes areas which are access protected during a user mode. The user and supervisor modes of operation are well known in the art and will not be discussed further. At step 62, the supervisor will first set the trace bit in, for example, bit field TR, the access protection bits, and the valid bit field V for each memory segment. At this point the supervisor has initialized the hardware in preparation for operation and user mode. Flow continues to step 64, where tracing of the access protected memory is enabled. Trace enablement indicates that access protect unit 20, as illustrated in FIG. 1, is enabled. Access protect unit 20 is enabled by the supervisor. Disabling access protect unit 20 bypasses tracing based on bit field TR. At step 66, operation of data processing system 10 is switched from supervisor mode to user mode. At step 68, an access to data processing system 10 is initiated.

Moughani, col. 3, l. 51 – col. 4, l. 4.

This section teaches the process of memory access tracing according to *Moughani*'s invention. One of the steps of *Moughani*'s process is to set a trace bit, among other bits, for each memory segment to be traced.

The Examiner equates this altering to the setting of the trace bit as described in the above section. However, note that *Moughani* expressly states that the <u>supervisor</u> sets the trace bit, <u>not a tracing function</u>,

such as the tracing function recited in claim 1. A supervisor is not the same as a tracing function. Therefore, a description of an action taken by a supervisor does not teach an action taken by a tracing function. Therefore, the cited section fails to teach the altering step of claim 1.

Therefore, *Moughani* fails to teach at least the receiving and altering steps as recited in claim 1. Consequently, the rejection of claim 1 under 35 U.S.C. § 102(b) has been overcome. Claims 4 and 7 contain features similar to those in claim 1, and are also not anticipated by *Moughani* by similar reasoning. Dependent claims 2-3, 5-6, and 8-9 are also not anticipated by *Moughani* at least by virtue of their dependence from claims 1, 4, and 7, respectively. Additionally, *Moughani* as a whole does not teach or suggest the above-described features of these claims to make these claims obvious.

I.B. <u>Claims 10, 12, and 14</u>

The Examiner has rejected claims 10 under 35 U.S.C. § 102(b) as being anticipated by *IBM technical disclosure bulletin NN9612331*, "Processor Single Step Trace Facility Enhancements", (published, December 1, 1996) (hereinafter, "*IBM*"). This rejection is respectfully traversed.

The Examiner has rejected this claim stating:

> As per claim 10, TDB NN96 123 1 discloses a method for reducing interrupts while tracing an application in a data processing system, the method comprising: initiating execution of tracing software (TDB NN96 123 1, first paragraph); allocating a data output buffer in physical memory, wherein the data output buffer holds output data from the tracing software (TDB NN96123 1, second paragraph, where the writing to the I/0 space is an access to physical memory); and writing output data to the data output buffer by the tracing software using physical memory addressing (TDB NN96123 1, second paragraph, where the 110 space is directly addressed).

Office Action dated November 17, 2004, p. 4.

Claim 10 recites:

> 10. A method for reducing interrupts while tracing an application in a data processing system, the method comprising:
> initiating execution of tracing software;
> allocating a data output buffer in physical memory, wherein the data output buffer holds output data from the tracing software; and
> writing output data to the data output buffer by the tracing software using physical memory addressing.

In the present case, each and every step in claim 10 is not shown in the cited reference. *IBM* does not teach the allocating step in the manner recited in claim 10. The Examiner cites the following section from *IBM* as teaching this feature of the claimed invention.

> The instruction/data tracing algorithm is enhanced v ia two methods: o First, more instruction decoding information is available at the time the interrupt is taken. o Second, instead of collecting trace data in a processor/memory buffer, the processor or the interrupt routine directly writes the data at a specified

> memory address, which could be defined in Input/Output (I/O) space. Data written to a specific address is written to the system bus. The data written to the specified address on the system bus is collected via a device attached to the system bus for future processing.

IBM, second paragraph. (typographical errors in the original).

In this section, *IBM* describes that instead of holding the interrupt data in a memory buffer, the interrupt routine writes the interrupt data directly to a memory address. Subsequently, data written to the memory address is written to the system bus. Devices connected via the system bus thereby have access to the data for further processing.

Note that the *IBM* reference states <u>instead of</u> collecting the trace data in a processor/memory buffer, meaning that the <u>reference does not desire collecting the data in a memory buffer</u>. In stark contrast, the claim recites allocating a data output buffer in physical memory wherein the data output buffer holds output data from the tracing software, meaning that the claimed step <u>does desire to collect the data in a memory buffer</u>. As a matter of express disclosure, *IBM* teaches away from the step recited in claim 10, rather than support the Examiner's contention. Therefore, *IBM* section as cited, or as a whole fails to teach the allocating step of claim 10.

Therefore, *IBM* fails to teach at least one feature as recited in claim 10. Consequently, the rejection of claim 10 under 35 U.S.C. § 102(b) has been overcome. Claims 12 and 14 contain features similar to those in claim 10, and are also not anticipated by *IBM* by similar reasoning. Additionally, *IBM* as a whole does not teach or suggest the above-described feature of these claims to make these claims obvious.

I.C. <u>Claims 11, 13, and 15</u>

The Examiner has rejected claims 11 under 35 U.S.C. § 102(b) as being anticipated by *IBM*. This rejection is respectfully traversed.

The Examiner has rejected this claim stating:

> As per claim 11, TDB NN96123 1 discloses a method for reducing interrupts while tracing an application in a data processing system, the method comprising: initiating execution of tracing s o b a r e (TDB NN96123 1, first paragraph); allocating a data output buffer, wherein the data output buffer holds output data from the tracing software (TDB NN96123 1, second paragraph, where an area of the I/O space is initialized); and configuring a translation register in a processor of the data processing system for the data output buffer (TDB NN96123 1, second paragraph, where the interrupt routine hnctions as a translation register by providing access to the specified memory address for the trace data that is transmitted, and translated to this memory space).

Office Action dated November 17, 2004, p. 4.

Claim 11 recites:

> 11. A method for reducing interrupts while tracing an application in a data processing system, the method comprising:
> initiating execution of tracing software;
> allocating a data output buffer, wherein the data output buffer holds output data from the tracing software; and
> configuring a translation register in a processor of the data processing system for the data output buffer.

In the present case, each and every step in claim 11 is not shown in the cited reference. *IBM* does not teach the allocating step in the manner recited in claim 11. Because the allocating step of claim 11 closely resembles the allocating feature of claim 10, the reasoning for this lack of teaching is the same as described above in section 1.B. Further, *IBM* also does not teach the configuring step of claim 11. The Examiner cites the same section from *IBM* as quoted above, as teaching this feature of the claimed invention. Quotation of that section and the description thereof are omitted here for the sake of brevity.

The Examiner concludes that the interrupt routine as described in the cited section functions as the translation register as recited in claim 11. However, the reference teaches no such function for the interrupt routine, and the conclusion drawn by the Examiner based on absent teaching is improper. Additionally, the reference teaches <u>an algorithm</u>, <u>not a routine</u>, and certainly <u>not a register</u>. To find merit in the Examiner's conclusion requires one to make several leaps of faith and read an algorithm described by the reference as a register for translating memory addresses. Therefore, the *IBM* section as cited, or as a whole, fails to teach the allocating and configuring steps of claim 11.

Therefore, *IBM* fails to teach at least two steps as recited in claim 11. Consequently, the rejection of claim 11 under 35 U.S.C. § 102(b) has been overcome. Claims 13 and 15 contain features similar to those in claim 11, and are also not anticipated by *IBM* by similar reasoning. Additionally, *IBM* as a whole does not teach or suggest the above-described feature of these claims to make these claims obvious.

I.D. <u>Claims 16-27</u>

The Examiner has rejected claims 16-27 under 35 U.S.C. § 102(b) as being anticipated by *Levine* et al., <u>Hardware Mechanism for Instruction/Data Address Tracing</u>, U.S. Patent No. 5,446,876 (issued, August 29, 1995) (hereinafter, "*Levine*"). This rejection is respectfully traversed.

The Examiner has rejected these claims stating:

> As per claim 16, Levine discloses a method for reducing interrupts while tracing an application in a data processing system, the method comprising: receiving an indication of an instruction to be traced, wherein the instruction is associated with an instruction address (Levine, col. 4, lines 51-59); in response to receiving the indication of the instruction to be traced, retrieving the instruction address; writing the instruction address to a trace output buffer in memory (Levine, col. 9, lines 10-1 1); and writing instruction resolution information to a trace output buffer, wherein the instruction resolution information comprises

> operating-system-defined memory allocation information or generated application code (Levine, col. 8, lines 25-43)

Office Action dated November 17, 2004, pp. 6-7.

Claim 16 recites:

> 16. A method for reducing interrupts while tracing an application in a data processing system, the method comprising:
>
> receiving an indication of an instruction to be traced, wherein the instruction is associated with an instruction address;
>
> in response to receiving the indication of the instruction to be traced, retrieving the instruction address;
>
> writing the instruction address to a trace output buffer in memory; and
>
> writing instruction resolution information to a trace output buffer, wherein the instruction resolution information comprises operating-system-defined memory allocation information or generated application code.

In the present case, each and every step in claim 16 is not shown in the cited reference. *Levine* does not teach the second writing step in the manner recited in claim 16. The Examiner cites the following section from *Levine* as teaching this feature of the claimed invention.

> Register 113 will contain a specific binary value indicating whether or not the effective to virtual address mapping scheme was changed by the previous instruction. This information is very important in a tracing context, since it may be necessary to use the physical address (data or instruction) to recreate the actual instruction that was executed, or data that was accessed. A mapping table, or look-up table is used to map a 32 bit effective address into a 52 bit virtual address. The actual physical address of the data or instruction is then determined from the virtual address. Periodically, instructions alter, or change, the content of the look-up table such that a particular effective address maps to a different virtual address. It is imperative that the tracing tool log the changes to the table occur when they occur. Otherwise, the actual data or instruction at a specified address may not be obtained by working backwards from the physical address to the virtual address to the effective address.

Levine, col. 8, ll. 25-43.

In this section, *Levine* describes a register to hold a binary value indicating whether or not an effective virtual address mapping scheme has changed. This flag is used in recreating the physical address from the virtual addresses stored in the address mapping table. *Levine* describes the effective addresses to be 32 bit long, and the virtual addresses to be 52 bit long. According to *Levine* the logging of the changes as they occur is imperative otherwise the address mapping will fail to work. Given this description, and the statement that a register is used to hold the indication of change, *Levine* teaches that any delay in writing the information will cause *Levine*'s method to fail. One of ordinary skill in the art knows that a register allows immediate read and write access to a small amount of data to other applications and devices that need that data.

In contrast, the claim recites a buffer to hold instruction resolution information. One of ordinary skill in the art also knows that a buffer by definition is something to hold data until the data can be written elsewhere where other applications and devices can make use of that data. Therefore, a register taught by *Levine* is not the same as a buffer as recited in claim 16.

Furthermore, because the binary value in *Levine* indicates a change in mapping scheme the change theoretically can be reflected in a binary value that is only 1-bit long. In any case, the binary value cannot be longer than the length of the register itself that holds the binary value.

This section in *Levine* fails to teach the second writing step as recited in claim 16. A register as used in the reference is not the same as a buffer, simply because of the size and purpose for which each item is used. The register in *Levine* holds a binary value only a few bits long. The buffer in the claim holds instruction resolution information, which can include memory allocation information or application codes, as the claim recites. That information can be much longer than a binary value to be storable in a register, as taught by *Levine*. Therefore, a register taught by *Levine* is not the same as a buffer as recited in claim 16 for this additional reason.

Therefore, *Levine* fails to teach at least the second writing step as recited in claim 16. Consequently, the rejection of claim 16 under 35 U.S.C. § 102(b) has been overcome. Claim 22 contains features similar to those in claim 16, and is also not anticipated by *Levine* by similar reasoning. Dependent claims 17-21 and 23-27 are also not anticipated by *Levine* at least by virtue of their dependence from claims 16 and 22, respectively. Additionally, *Levine* as a whole does not teach or suggest the above-described feature of these claims to make these claims obvious.

II. 35 U.S.C. § 103, Obviousness

The Examiner has rejected claims 28-33, and 40-42 under 35 U.S.C. § 103(a) as being unpatentable over *Levine*, in view of *Tanenbaum*, Structured Computer Organization, (published 1990) (hereinafter, "*Tanenbaum*"). This rejection is respectfully traversed.

The Examiner has rejected claim 28 stating:

> As per claim 28, Levine discloses a method for tracing comprising: means for receiving an indication of an instruction to be traced, wherein the instruction is associated with an instruction address (Levine, col. 4, lines 5 1-59); means for retrieving the instruction address in response to receiving the indication of the instruction to be traced; means for writing the instruction address to a trace output buffer in memory (Levine, col. 9, lines 10-1 1); and means for writing instruction resolution information to a trace output buffer, wherein the instruction resolution information comprises operating-system-defined memory allocation information or generated application code (Levine, col. 8, lines 25-43). Levine fails to disclose this method being implemented using a computer program product in a computer-readable medium.

Office Action dated November 17, 2004, pp. 6-7.

II.A. <u>The Cited References Do Not Teach all of the Features of Claims 28-33</u>

The Examiner has failed to state a *prima facie* obviousness rejection because the cited references used in proposed combination do not teach all of the features of claim 28 as believed by the Examiner.

As shown in section 1.D. above, *Levine* fails to teach at least the second writing step as recited in claim 16. Claim 28 contains features similar to those in claim 16, and is also not anticipated by *Levine* by similar reasoning. Dependent claims 29-33 are also not anticipated by *Levine* at least by virtue of their dependence from claim 28.

The Examiner has cited sections from *Tanenbaum* only for finding a teaching of a computer program product being in a computer readable medium. *Tanenbaum* has not been cited to teach or suggest any of the deficiencies described in *Levine* in section 1.D. above. Even if, *arguendo, Tanenbaum* teaches all that the Examiner alleges it teaches, the combination of *Levine* and *Tanenbaum* fails to teach or suggest all the features of claim 28 by the reasoning advanced above.

III. <u>Objection to claims 2-3, 5-6, and 8-9</u>

The Examiner has objected to claims 2-3, 5-6, and 8-9 as being dependent upon a rejected base claim. Claims 2, 5, and 8 have been amended accordingly and claims 2-3, 5-6, and 8-9 are now in the condition for allowance.

IV. **Conclusion**

It is respectfully urged that the subject application is patentable over *Moughani, IBM, Levine, and Tanenbaum,* and is now in condition for allowance.

The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: December 15, 2006

Respectfully submitted,

/Rakesh Garg/
Rakesh Garg
Reg. No. 57,434
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Agent for Applicant